Parveen Khurana

124 Followers About Following

Sigmoid Neuron

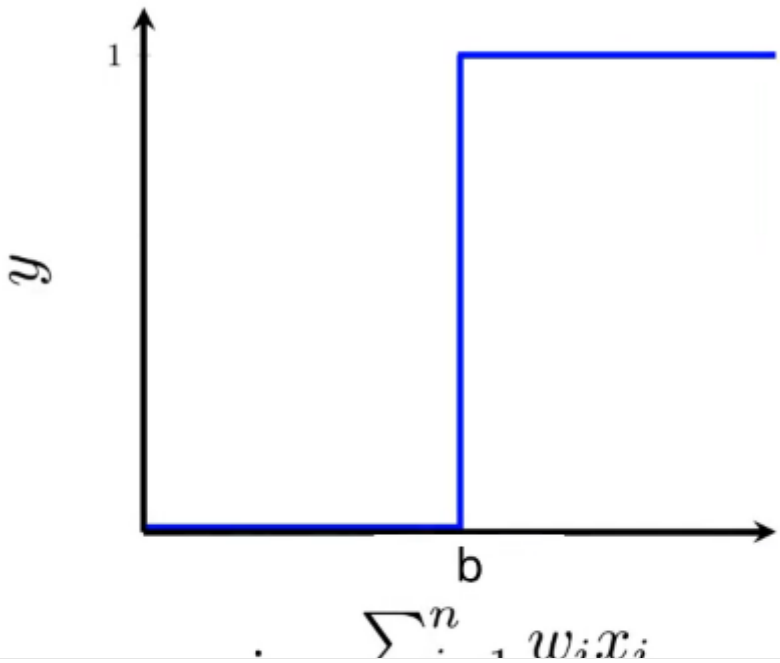
P Parveen Khurana Jan 3, 2020 · 13 min read

This article covers the content discussed in the Sigmoid Neuron module of the Deep Learning course and all the images are taken from the same module.

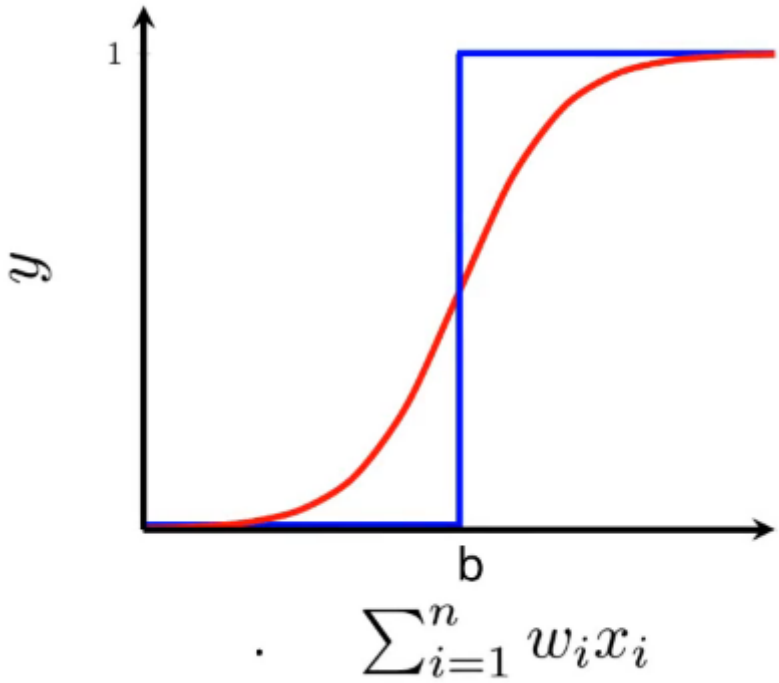
In this article, we discuss the 6 jars of the Machine Learning with respect to the Sigmoid Model but before beginning with that let's see the drawback of the Perceptron Model.

Sigmoid Model and a drawback of the Perceptron Model:

The limitation of the perceptron model is that we have this harsh function(boundary) separating the classes on two sides as depicted below



And we would like to have a smoother transition curve which is closer to the way humans make decisions in the sense that something is not drastically changed, it slowly changes over a range of values. So, we would like to have something like the **S-shaped** function(in red in the below image).

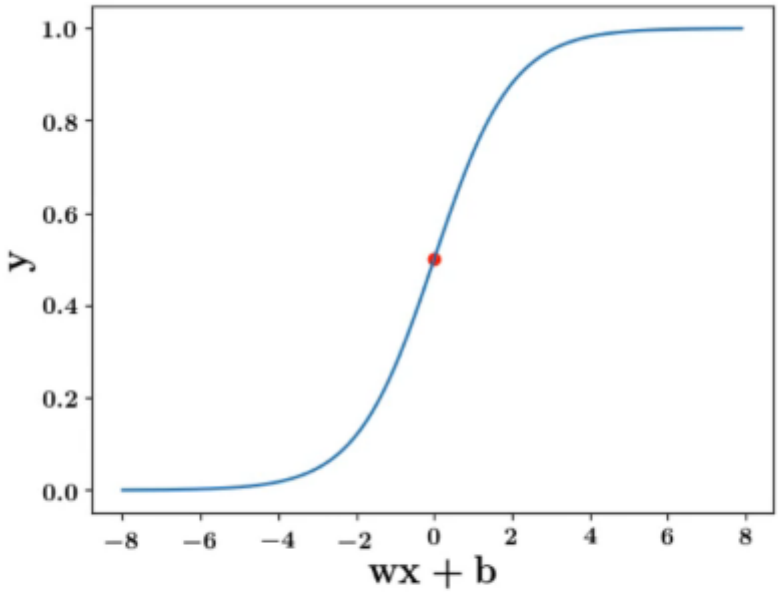


And we have the Sigmoid family of functions in Deep Learning of which many of the functions are S-shaped. One such function is the **logistic function**(it is one smooth continuous function) and this function **is defined by the below equation**:

$$y = \frac{1}{1 + e^{-(wx+b)}}$$

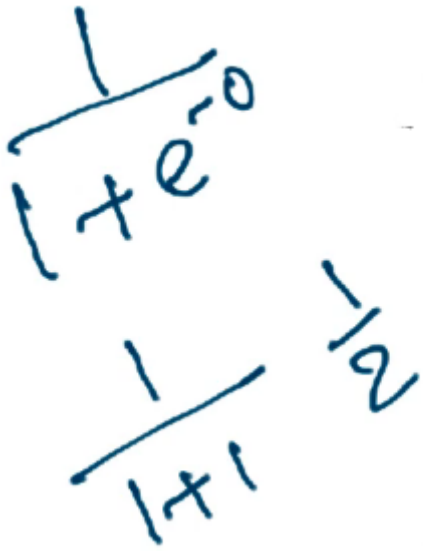
So, **we will now approximate the relationship between the input x**(which could be n-dimensional) **and the output y using this Logistic function**(Sigmoid function). This function would have some parameters and we would try to learn the parameters using the data in such a way that the loss is minimized.

axis and ‘y’ value on the y-axis.

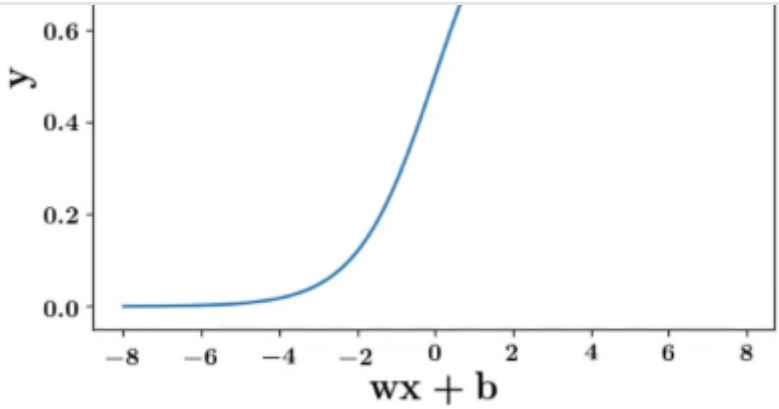


$$wx + b = 0, y = 0.5$$

If ‘**wx + b**’ is 0, then the equation(**y**) is reduced to:



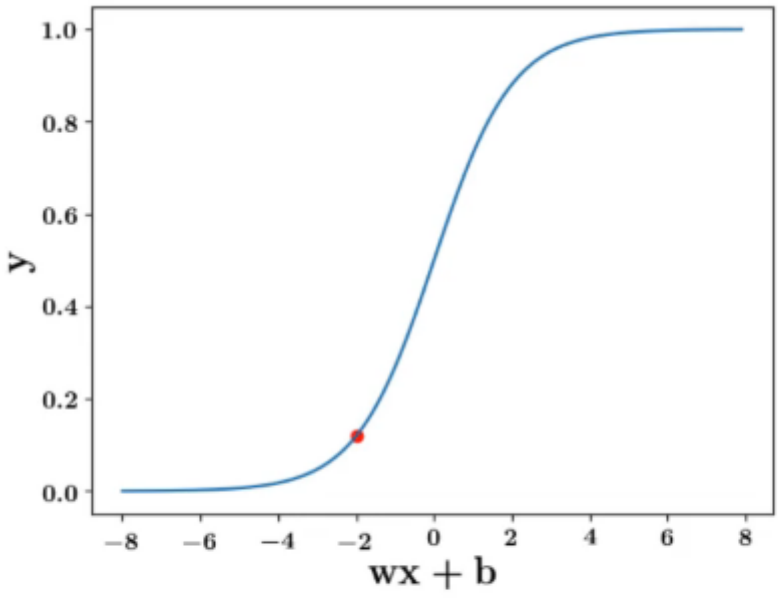
Let’s try some other value:



$$wx + b = 2, y = 0.88$$

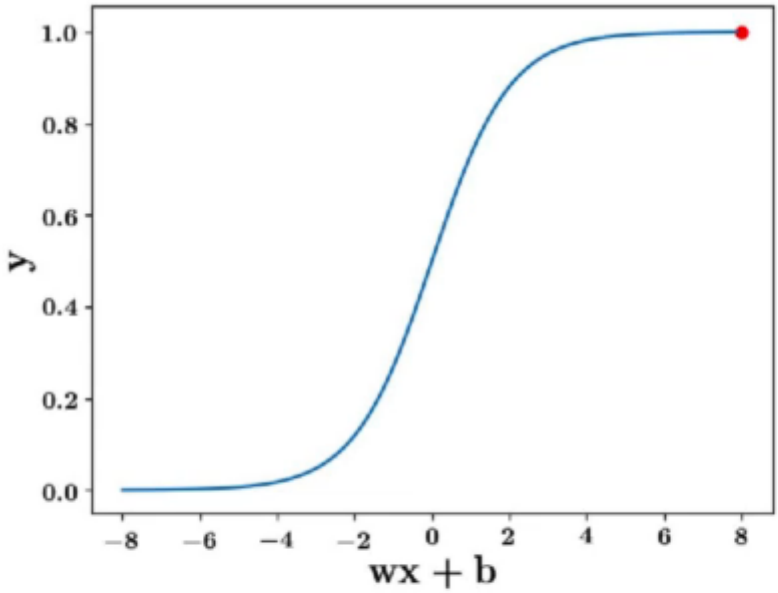
And y, in this case, would be:

$$\frac{1}{1 + e^{-2}}$$

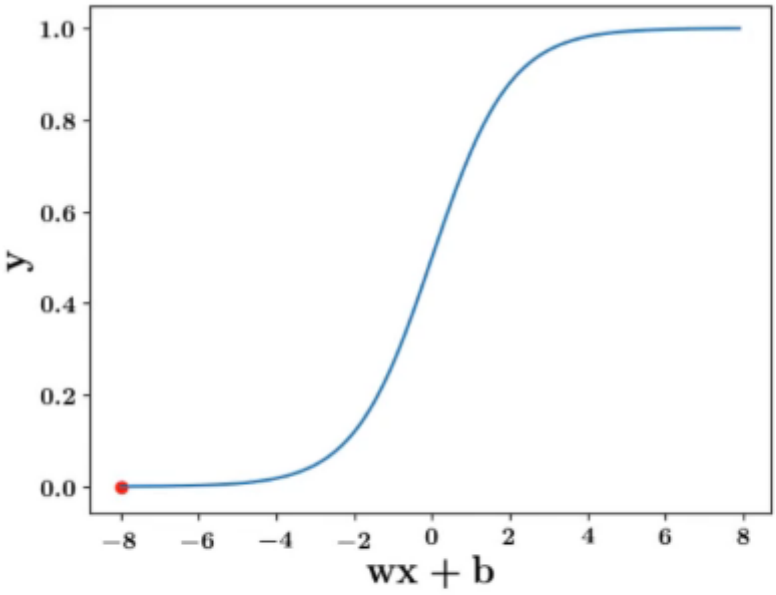


$$wx + b = -2, y = 0.12$$

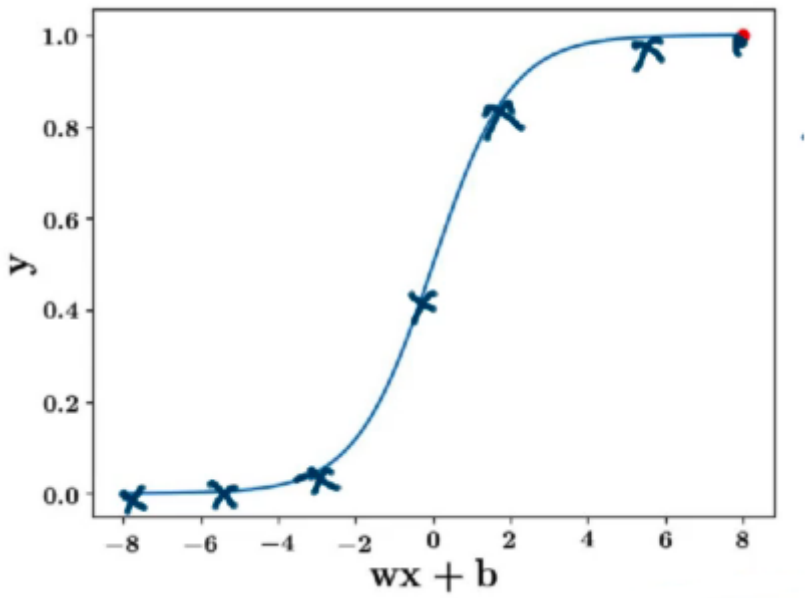
$$\frac{1}{1+e^{z}}$$



$$wx + b = 8, y = 0.9997$$



$$wx + b = -8, y = 0.0003$$



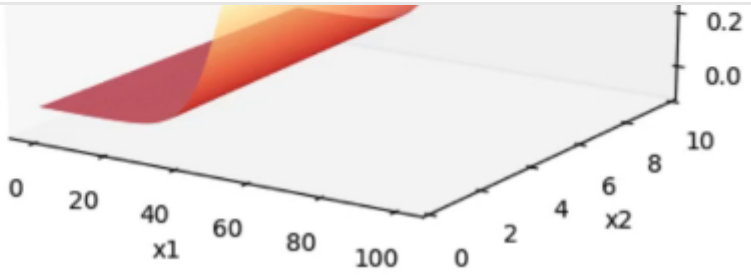
And we can get the general trend of the function. So, **we can visualize the function to see what the function looks like or how it varies with respect to the input.**

So, this is clearly is a smoother function as opposed to the if-else condition that we have in the Perceptron case.

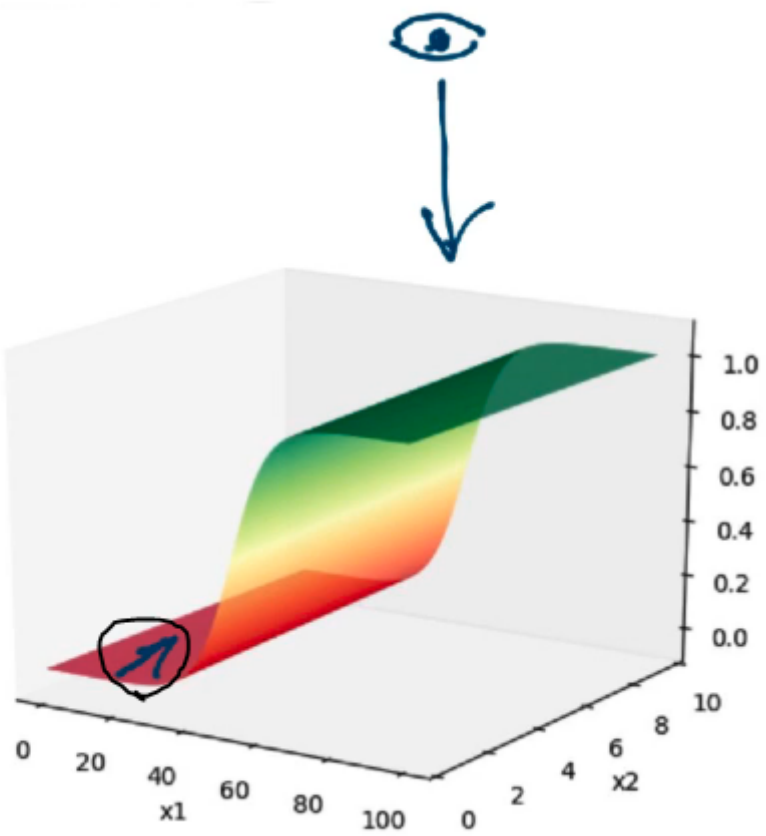
For the 2 inputs case, the function equation would be:

$$y = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + b)}}$$

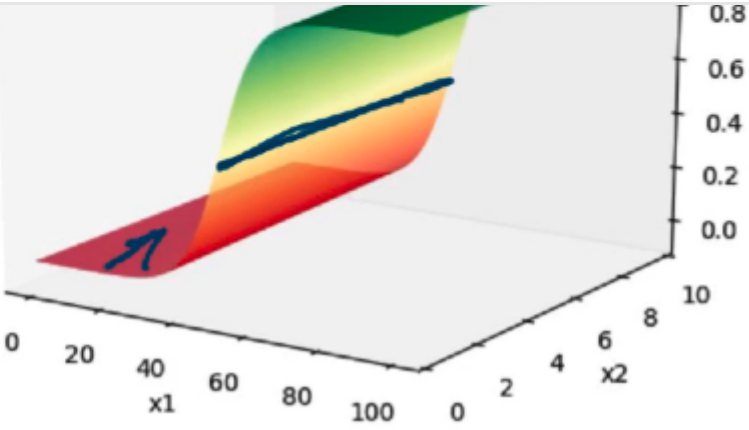
And if we plot it, it would look like:



To understand the plot better, we try to look at it from the top:



The dark red region(circled) in the above image is the region where the output value is close to 0 because as we plug larger and larger values, the output would be close to 0.



The green region would correspond to value 1 and the middle region(orange color) would correspond to value 0.5.

If we have more than 2 inputs, then we would write our equation as:

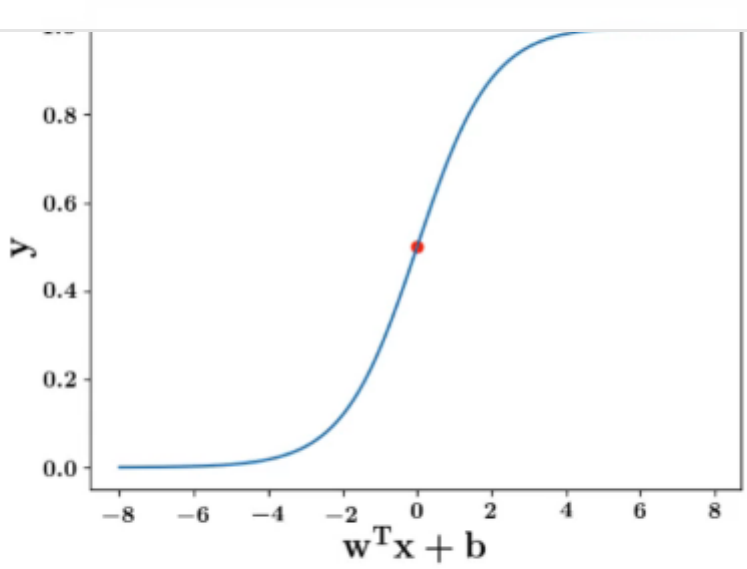
$$y = \frac{1}{1+e^{-(\sum_i w_i x_i + b)}}$$

And the below summation

$$\sum_i w_i x_i$$

would be the same as the dot product of the two vectors w and x.

$$y = \frac{1}{1+e^{-(w^T x + b)}}$$



$$w^T x + b = 0, y = 0.5$$

The output is going to be a scalar value between 0 and 1 no matter how many inputs we have.

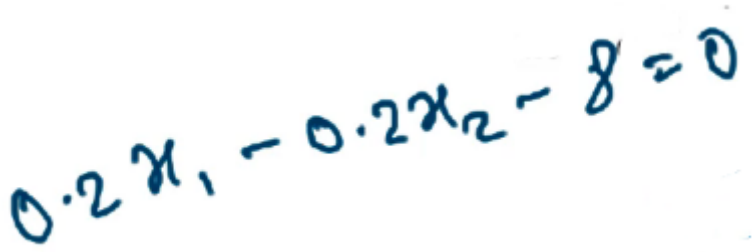
Let's consider the 2-D case, we have the equation as:

$$y = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + b)}}$$

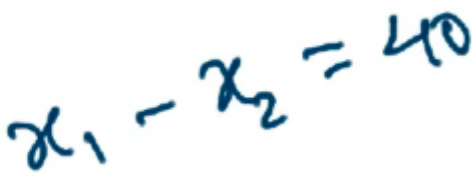
Let the value of **w1 be 0.2**, **w2 be -0.2** and **b be equal to -8**.

The output of the sigmoid function would be equal to 0.5 when the below quantity is 0 as then only the overall denominator would be 2:

Putting in the values of w1, w2, and b:



which is same as



So, **for this 2D case**, **whenever the difference of the two input values is 40**, **then the sum w1x1 + w2x2 +b would be 0 and in effect, the value y would be 0.5**. And this is how we can go about plotting out this function.

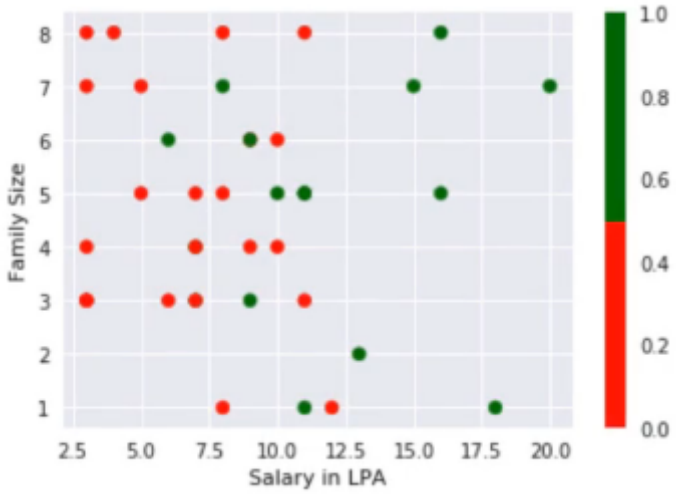
How does the model help when the data is not linearly separable?

Let's consider the below case where we have two inputs: Salary in LPA and Family Size and based on these inputs, we are going to make a decision whether that person is going to buy a car or not. We are assuming that there is some relation between the inputs(**x1 and x2**) and the output **y**. We don't know the true relation between the input and the output and we are approximating this relation using the sigmoid(logistic) function.

$$y = \frac{1}{1+e^{-(w^T x+b)}}$$

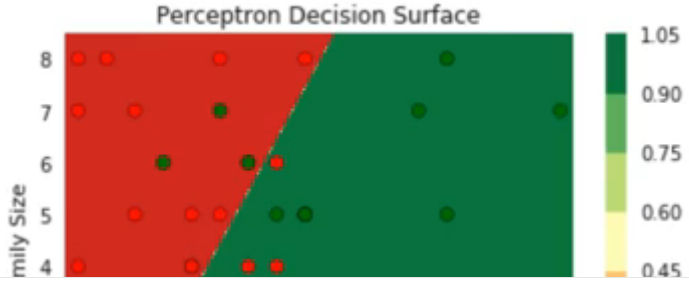
	Salary in LPA	Family Size	Buys Car?
0	11	8	1
1	20	7	1
2	4	8	0
3	8	7	0
4	11	5	1

If we plot out all the data:



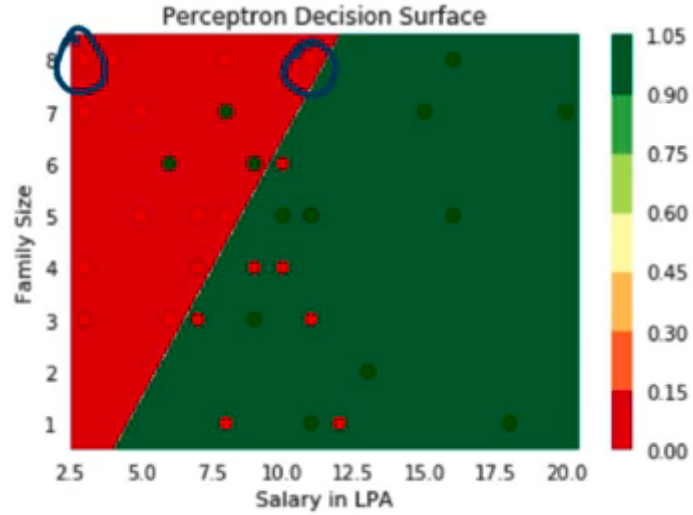
Red points are the points for which the output is 0 and the green points are the ones for which the output is 1. And it is clear that no matter how we draw a line we would not be able to separate the red points from the green points. And if train a perceptron model on this, it would not converge for sure but we would train it in a way such that we are okay with the number of errors it makes(meaning the wrong classification of some points).

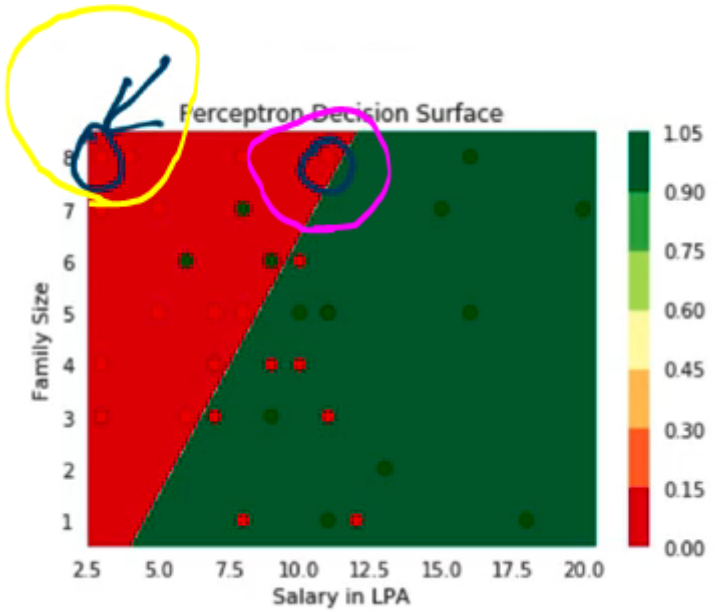
And if we plot out the perceptron linear boundary for the above data, we have:



In the above image, in the red region, we largely have the red points and in the green region, we largely have the green points but off course, there is an error on both sides.

The important thing to note is that the perceptron does not make any distinction between the two circled points in the below image:





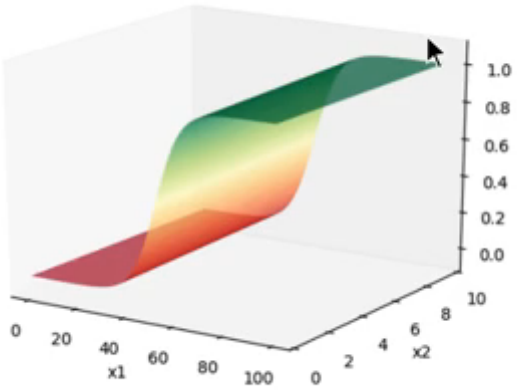
and of family size 8, a human decision-maker would be very confident that this person will not buy a car whereas, for the point in pink in the above image, we would be slightly confused whether this person may buy or may not buy a car. But if we look at the Perceptron decision boundary, its very firm meaning the model is confident for both the points(in yellow and in pink in the above image) that the person is not going to buy a car even though there is difference between these two points; one is near the boundary almost sitting at the fence whereas the other one is way inside the boundary but the Perceptron decision surface or the perceptron output is not able to make these distinctions because the output is either 1 or 0, it's not a smooth number between 0 to 1.

Now let's see what would be the scenario if we try to fit it using the Sigmoid function:

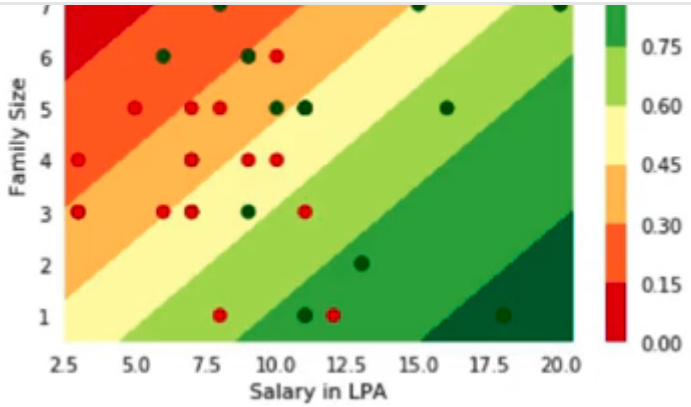
We will look at the data and using some learning algorithm and some loss function, we will find the parameters of the model/function.

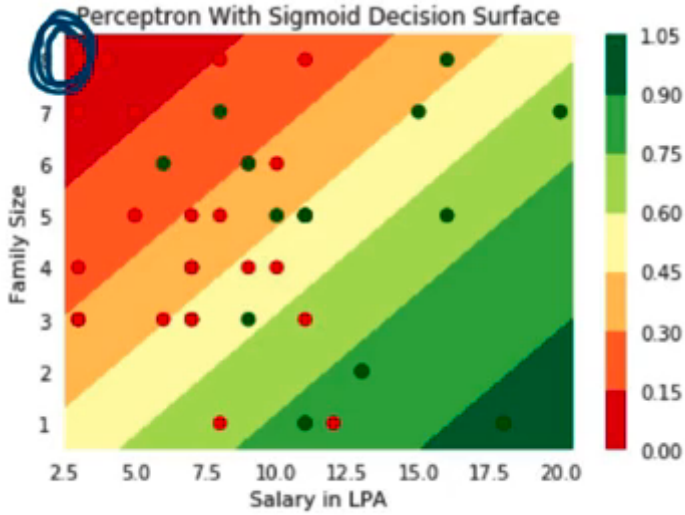
$$y = \frac{1}{1+e^{-(w^T x+b)}}$$

If we try to fit the data using Sigmoid, we would get the below kind of plot:



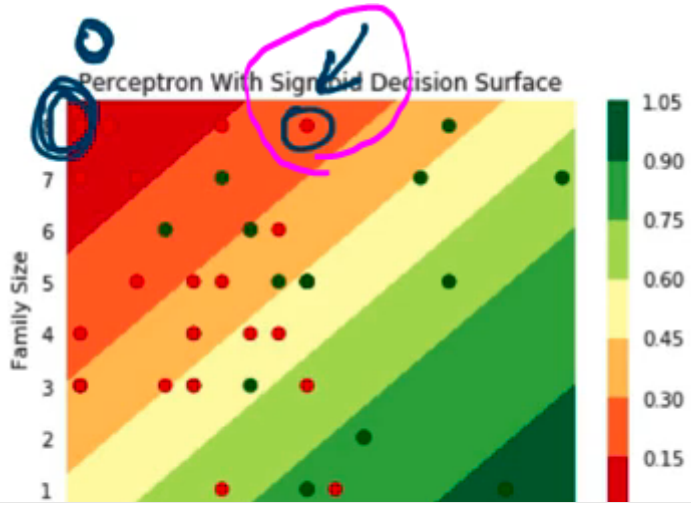
And the equivalent 2D plot would like:





If we look at the circled points in the above image, for a person with an annual income of 2.5 Lakhs and with a family size of 8, the output is close to 0(as the point lies in the dark red region for which the output is 0 or close to 0).

And for the pink circled input point in the below image

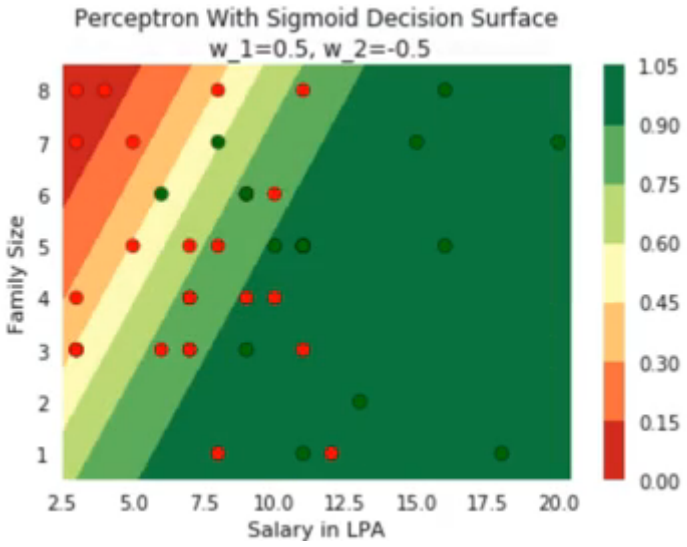


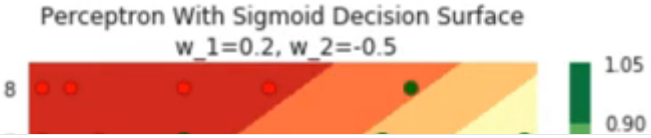
the output would be close to 0.3 or 0.4 which means the model is not very confident, it thinks it is on the lower side, it's not clearly 1, its not clearly 0 but maybe on the lower side, there is 30% chance that this person might buy a car. **So, that's the interesting thing about Sigmoid function, since it lies between 0 and 1 and another quantity of interest that we care about is Probability which also lies between 0 and 1. So, we can actually interpret the output of the Sigmoid Neuron as a probability. So, when it is 0, we can say there is a 0% chance of this person buying a car and the output of Sigmoid is 1 we can say there is a 100% chance of this person buying a car and so on.**

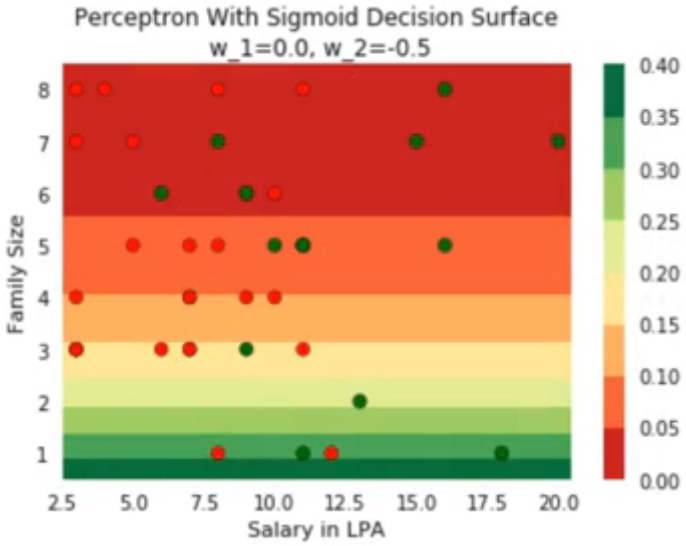
So, now we have this nice way of interpreting the output rather than being very rigid which means saying 0 here and 1 here, we can also account for the fence-sitters and we can say that this person is on leaning towards the positive side but completely towards 1. So, this is how we can actually interpret the output of the sigmoid function.

We are still not able to separate the green points from the red points. The non-linearity that we have introduced is giving us a graded output which allows a better interpretation to evaluate it in terms of probability.

Now, as we keep changing the values of the parameters w, b, we will get different types of sigmoid function for example:



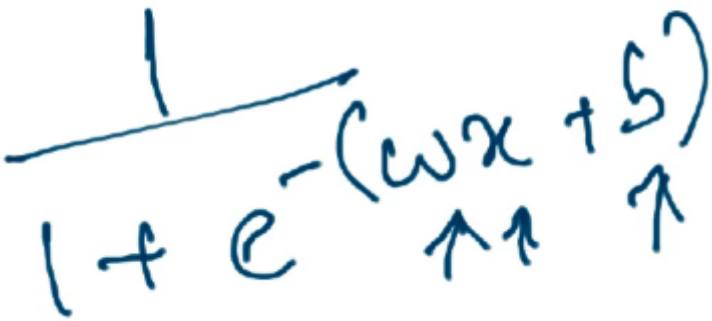




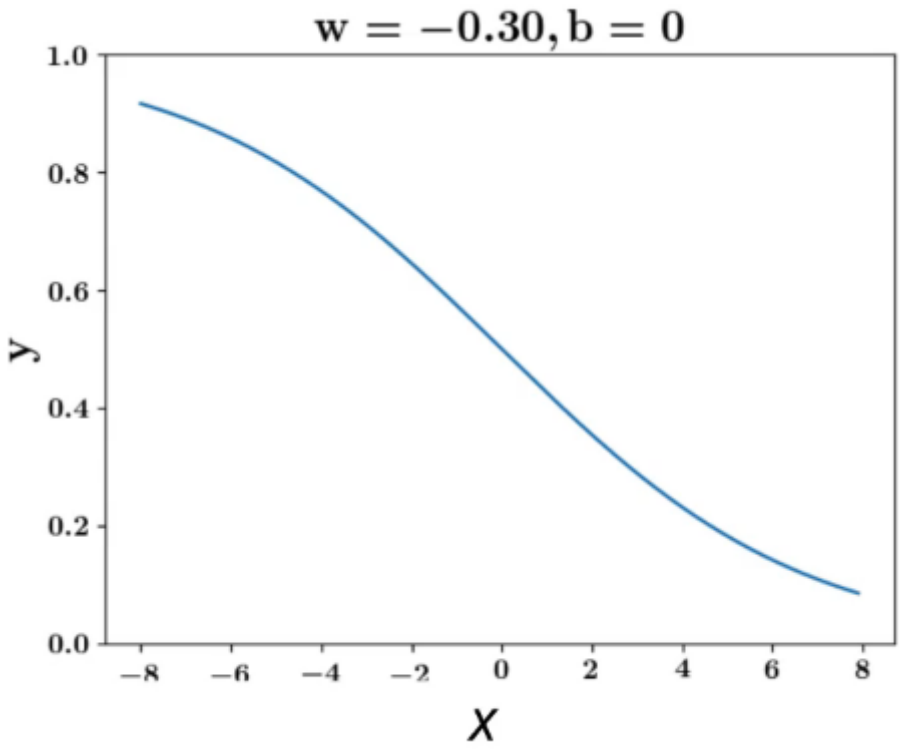
We will get different sigmoid plots for different values of the parameters but none of them would be able to separate the green points from the redpoint.

How does the function change with the change in w and b?

Let's consider this for only one input, in that case, we have the equation as:

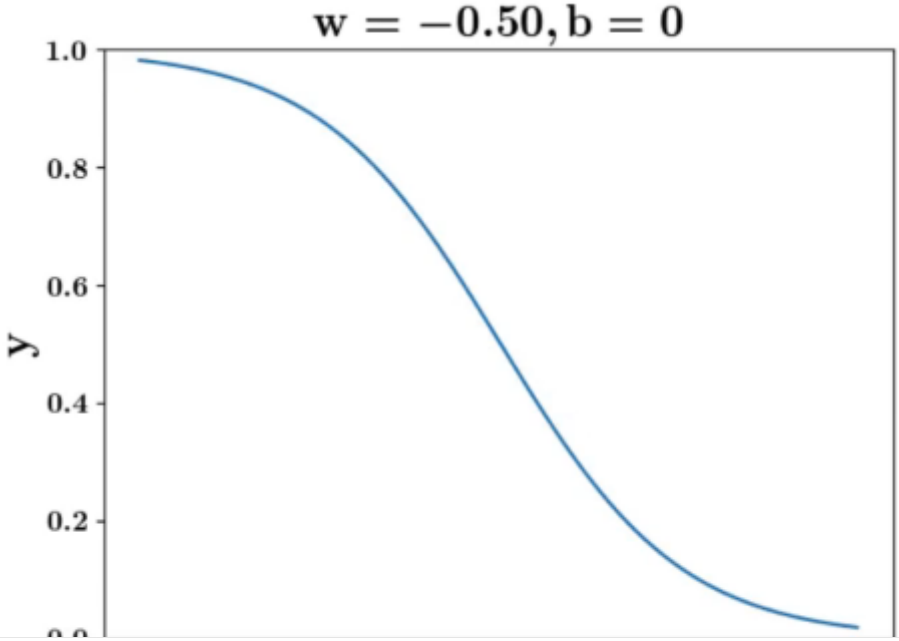


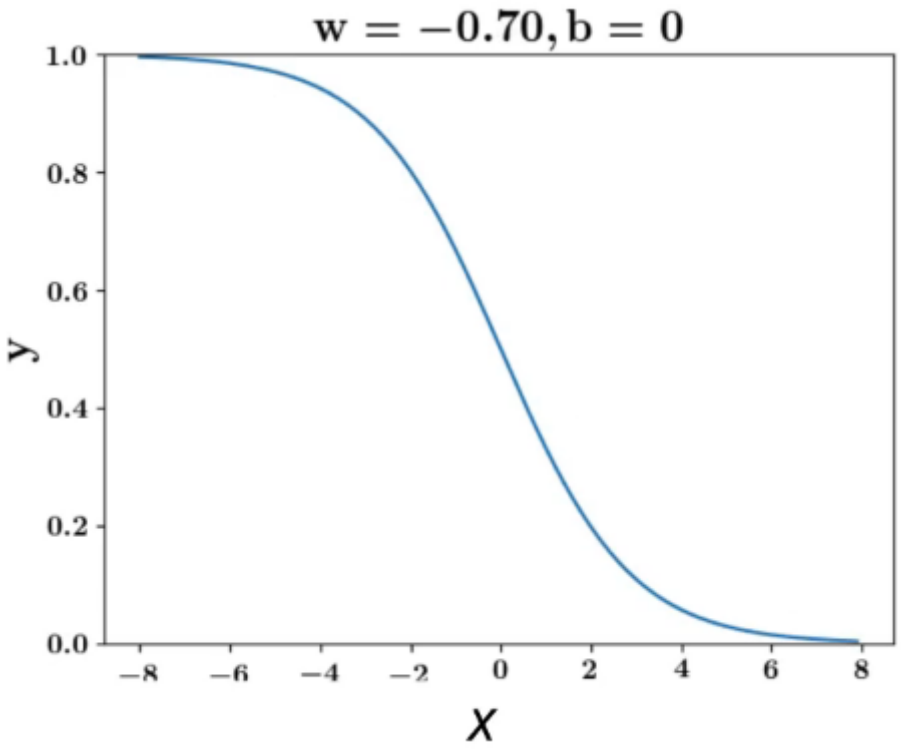
where the parameters **w and b** are going to be a scalar value and **x** represents the input.

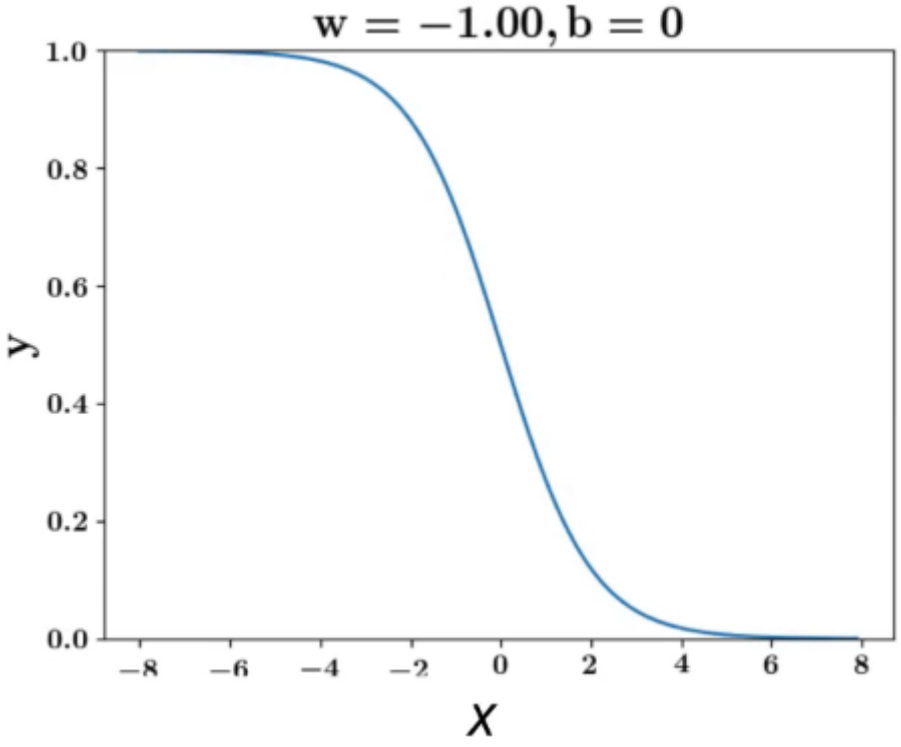


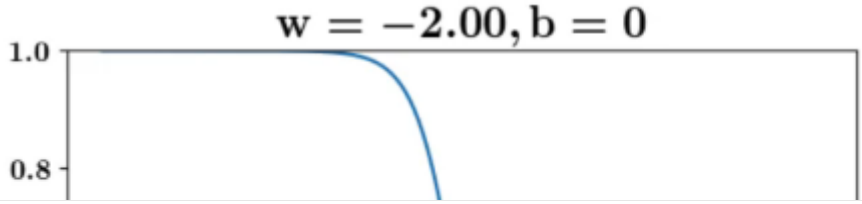
As w is negative, the slope of the sigmoid function is also negative, so what is happening is that as we are increasing the values of x, the value of the output is decreasing, this is what the negative slope means.

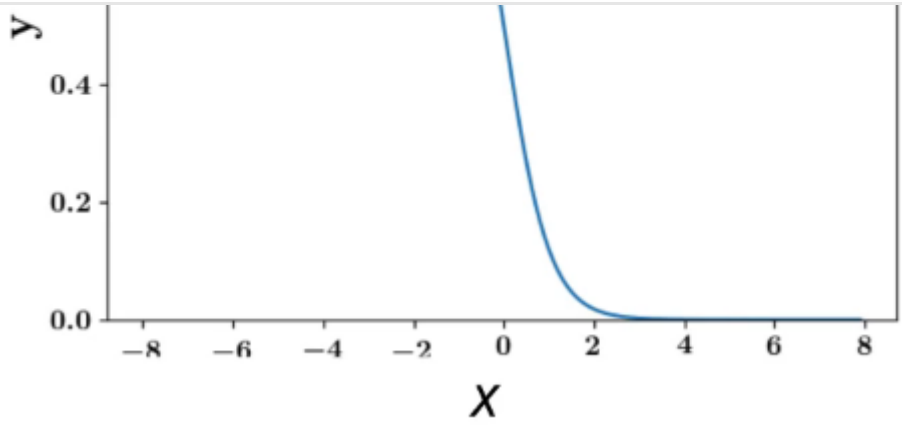
And as we keep increasing the slope or rather make it more and more negative, the curve becomes sharper, **that's what a high negative slope means, even if we change the value of x slightly, the value of the output is dropping drastically**:

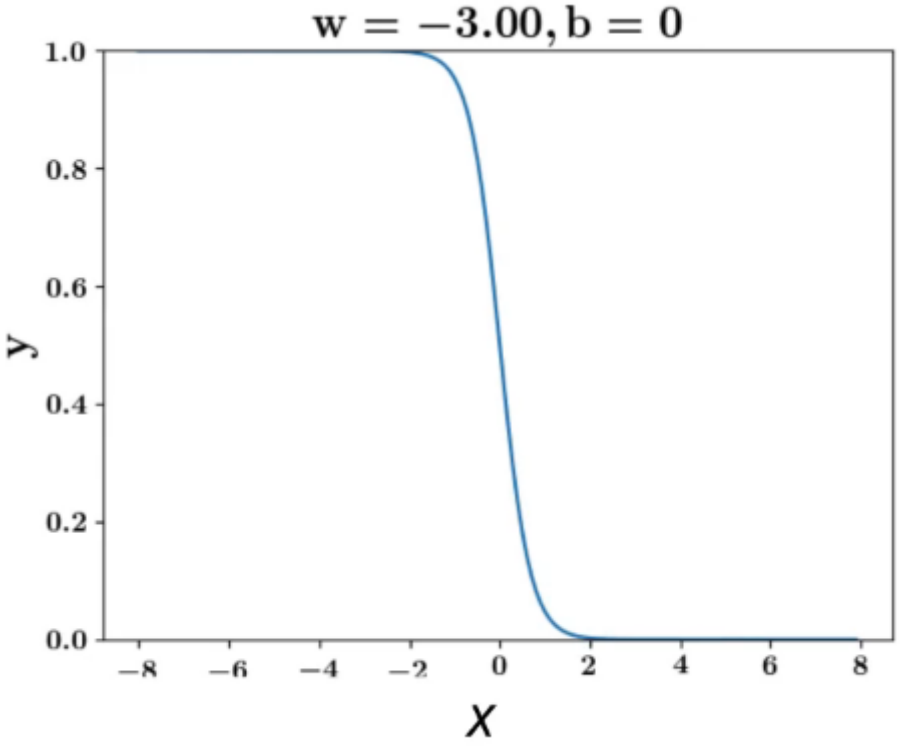


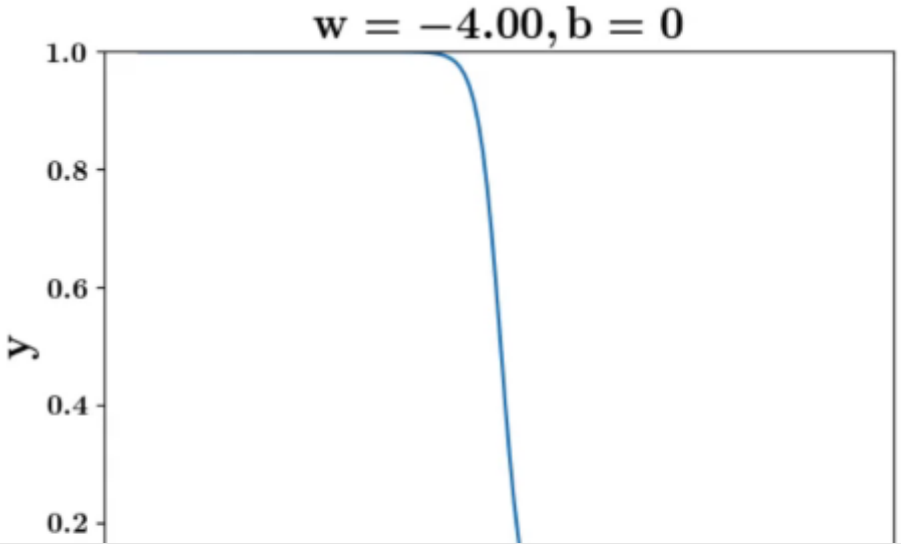

w = −0.70, b = 0
1.0
0.8
0.6
0.4
0.2
0.0
y
−8
−6
−4
−2
0
2
4
6
8
X

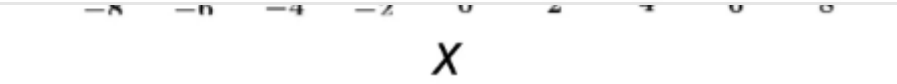

w = −1.00, b = 0
1.0
0.8
0.6
0.4
0.2
0.0
y
−8
−6
−4
−2
0
2
4
6
8
X

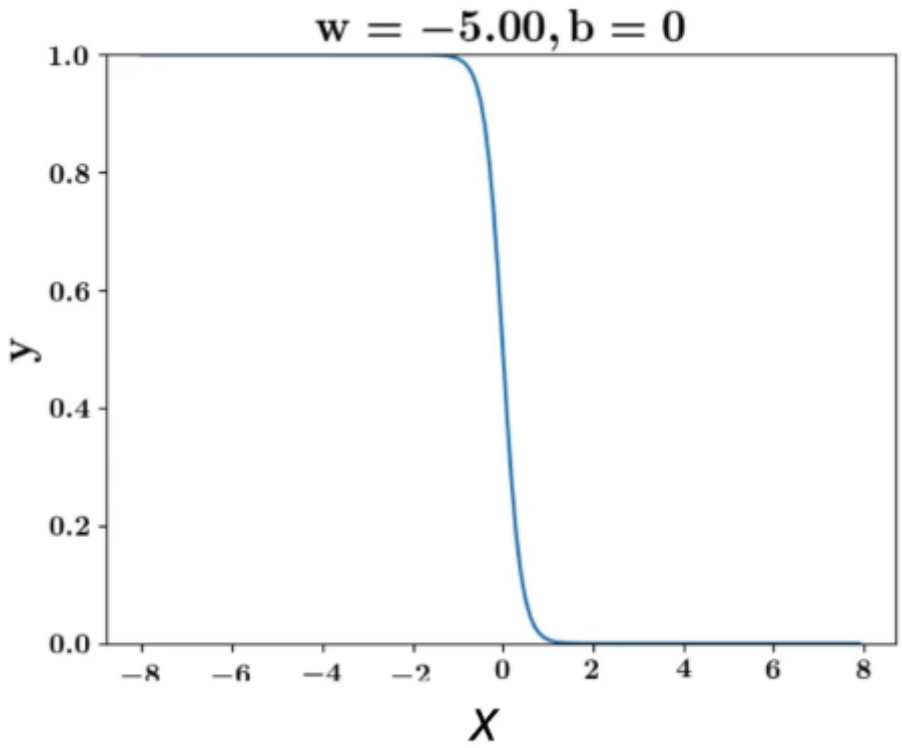

w = −2.00, b = 0
1.0
0.8


y
0.4
0.2
0.0
−8
−6
−4
−2
0
2
4
6
8
X

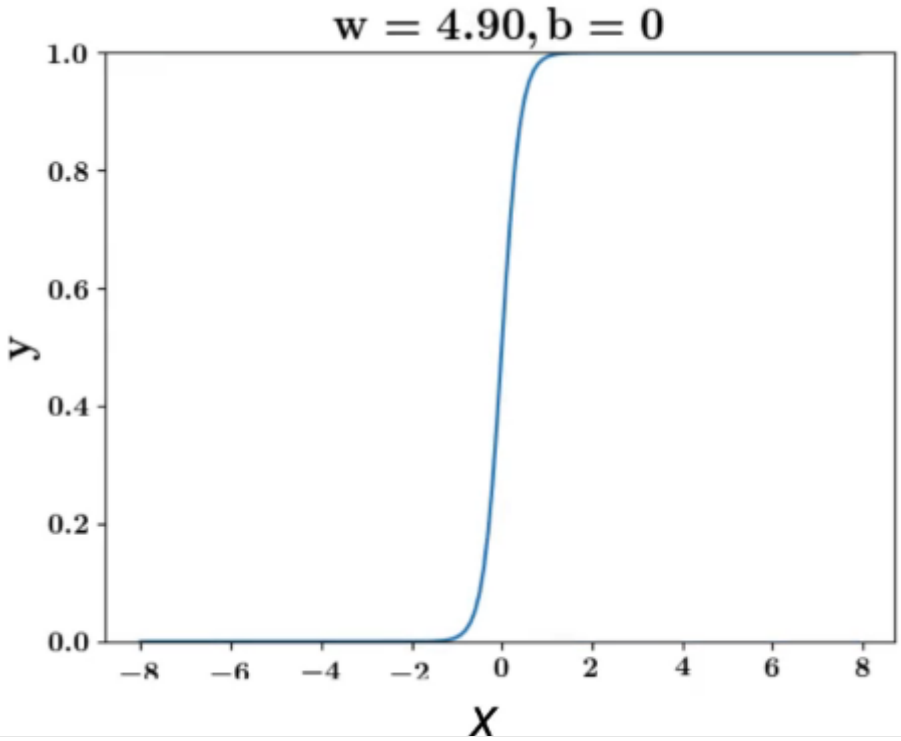

w = −3.00, b = 0
1.0
0.8
0.6
y
0.4
0.2
0.0
−8
−6
−4
−2
0
2
4
6
8
X

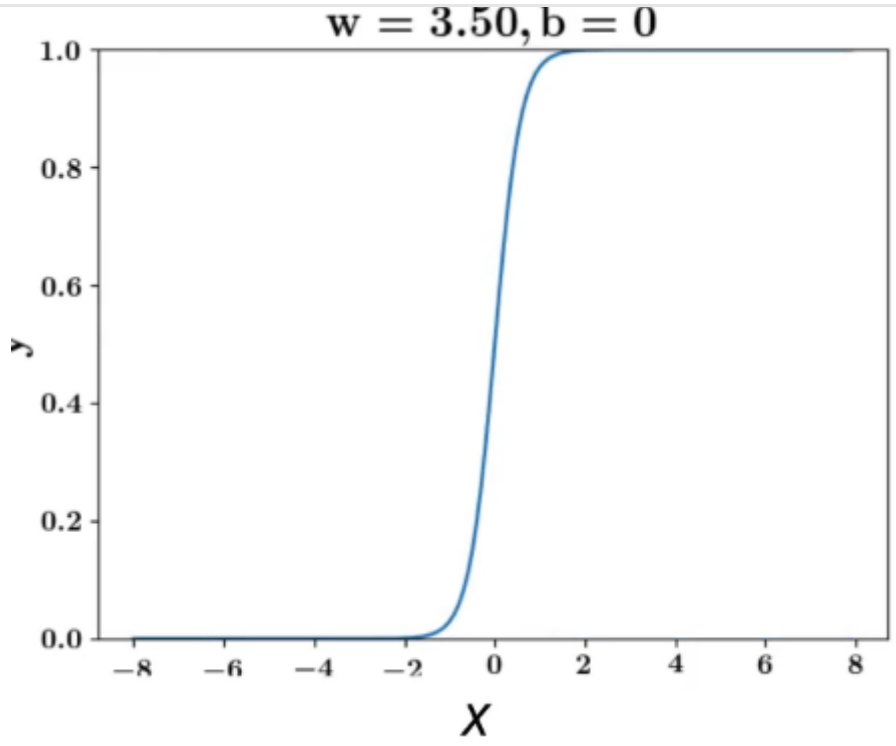

w = −4.00, b = 0
1.0
0.8
0.6
y
0.4
0.2

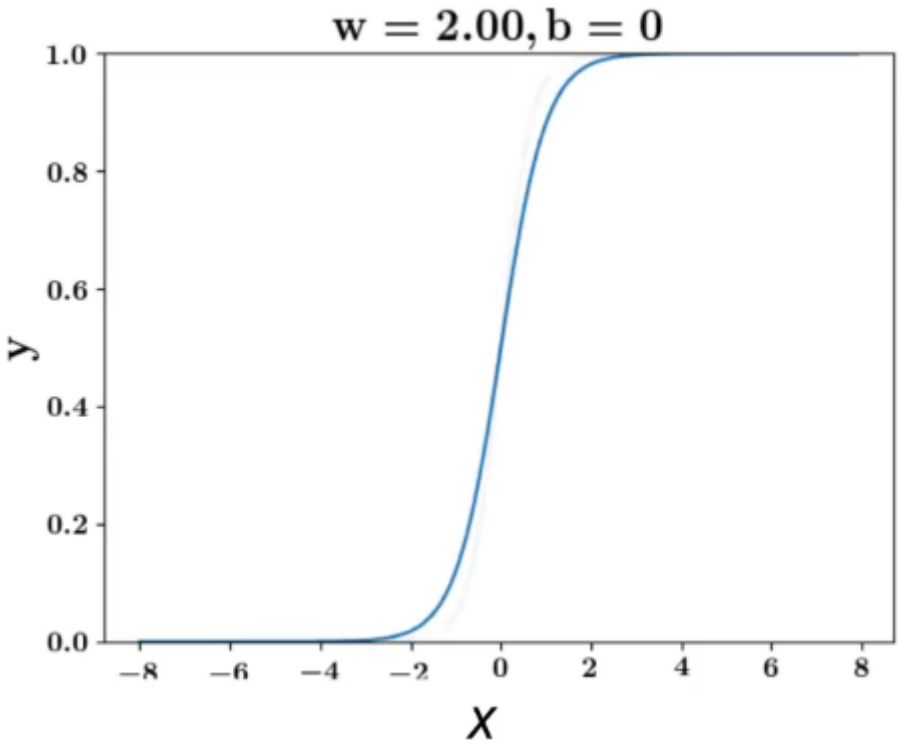


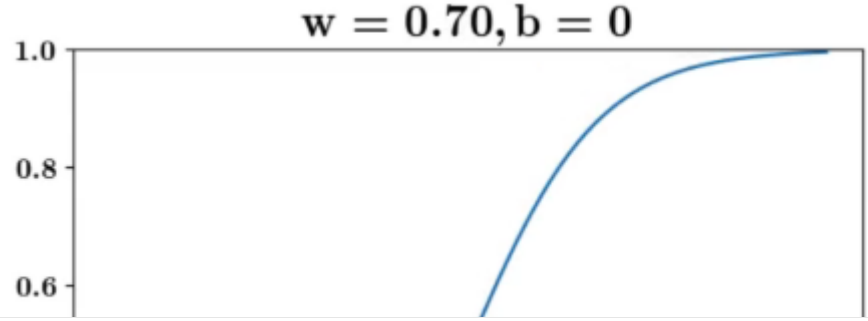


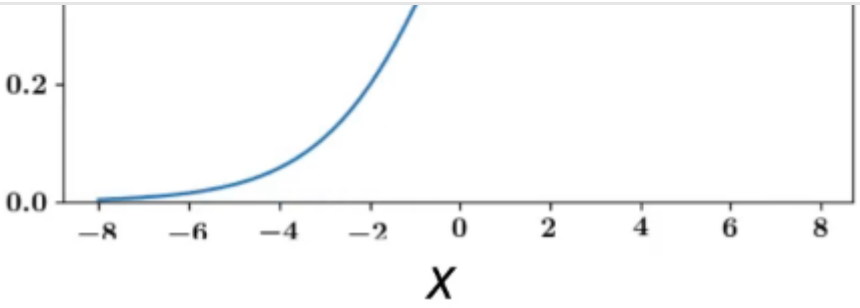
And now if we make the value of w positive, **the slope is going to be positive and the smaller the slope the less drastic is the change in the value of output.**

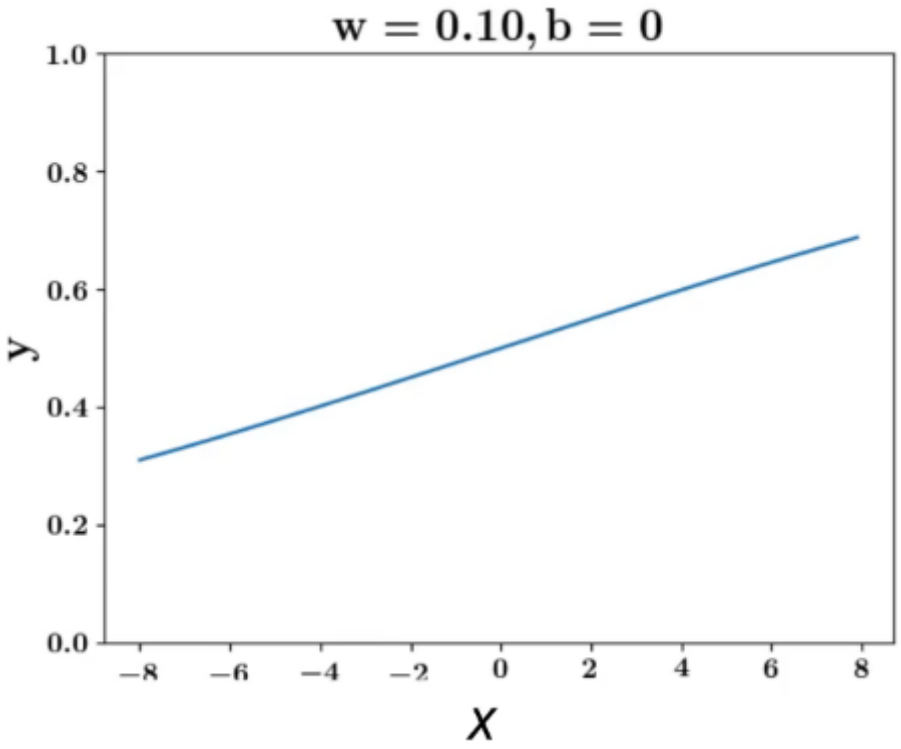



w = 3.50, b = 0
1.0
0.8
0.6
0.4
0.2
0.0
y
−8
−6
−4
−2
0
2
4
6
8
X


w = 2.00, b = 0
1.0
0.8
0.6
0.4
0.2
0.0
y
−8
−6
−4
−2
0
2
4
6
8
X

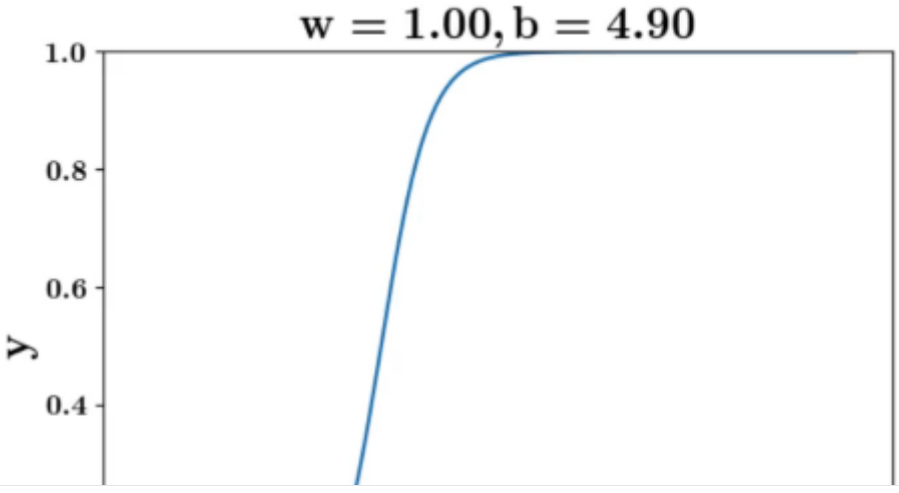

w = 0.70, b = 0
1.0
0.8
0.6

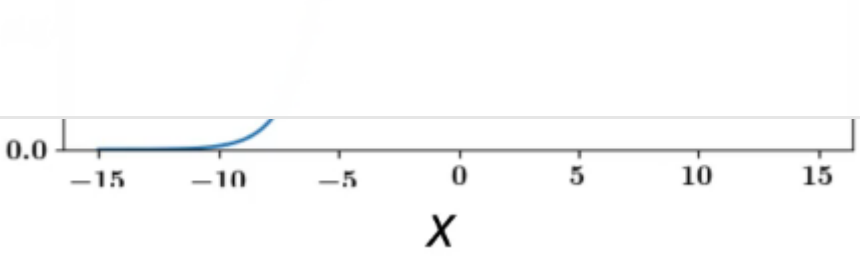


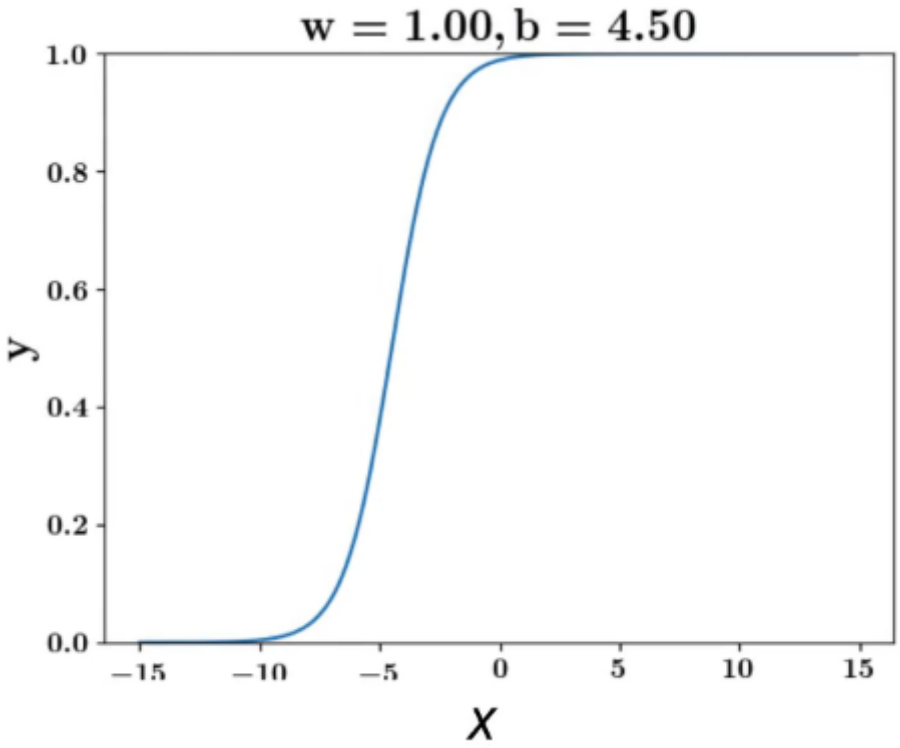


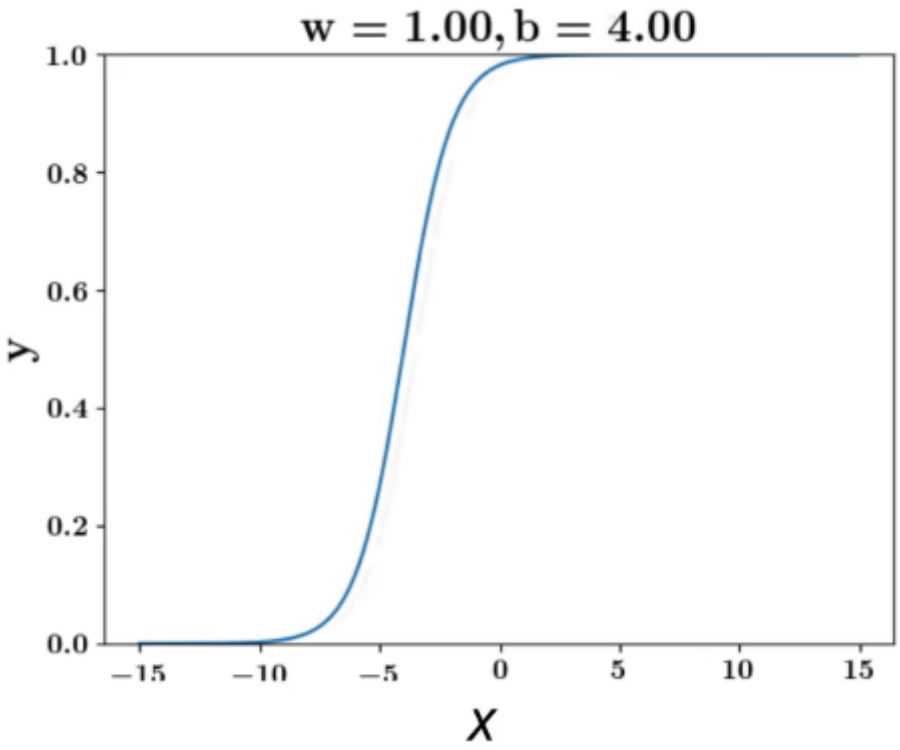
And the next thing to show is how the function change as we change the value of b:

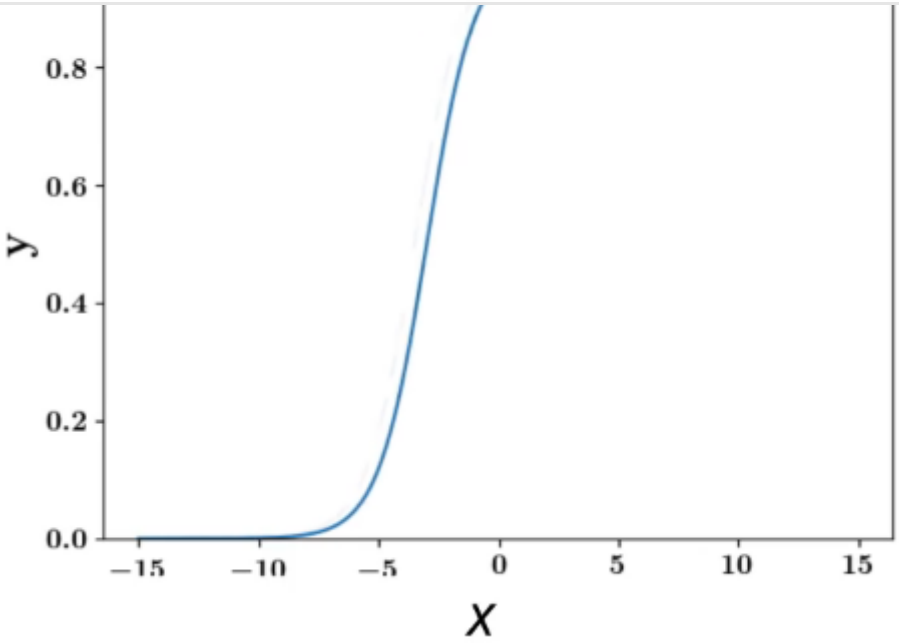
To start with, we have taken the value of b as 4.9 and **if we keep decreasing the value of b(keeping w constant), the function would shift towards the right**.

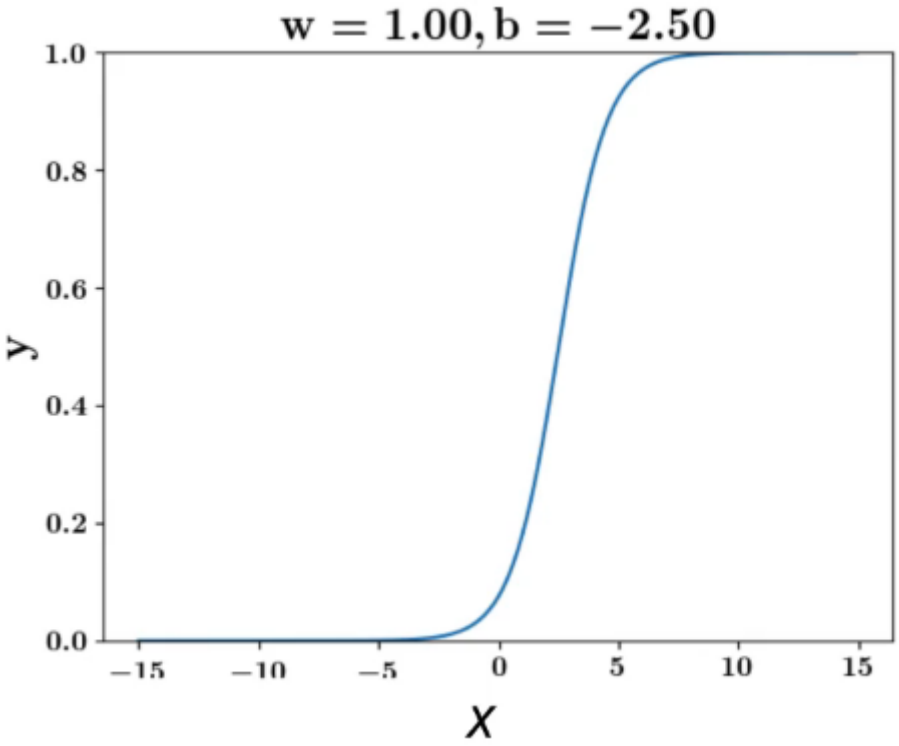


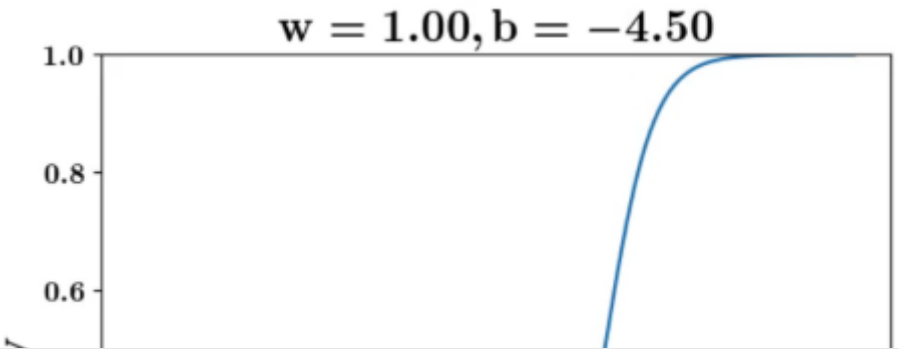

0.0
−15
−10
−5
0
5
10
15
X

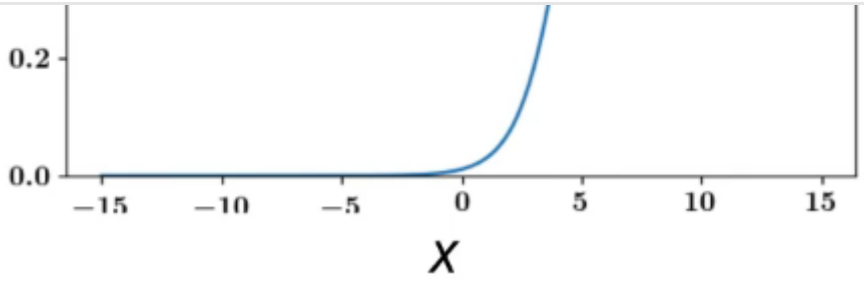

w = 1.00, b = 4.50
1.0
0.8
0.6
0.4
0.2
0.0
y
−15
−10
−5
0
5
10
15
X


w = 1.00, b = 4.00
1.0
0.8
0.6
0.4
0.2
0.0
y
−15
−10
−5
0
5
10
15
X


0.8
0.6
y
0.4
0.2
0.0
−15
−10
−5
0
5
10
15
X


w = 1.00, b = −2.50
1.0
0.8
0.6
y
0.4
0.2
0.0
−15
−10
−5
0
5
10
15
X

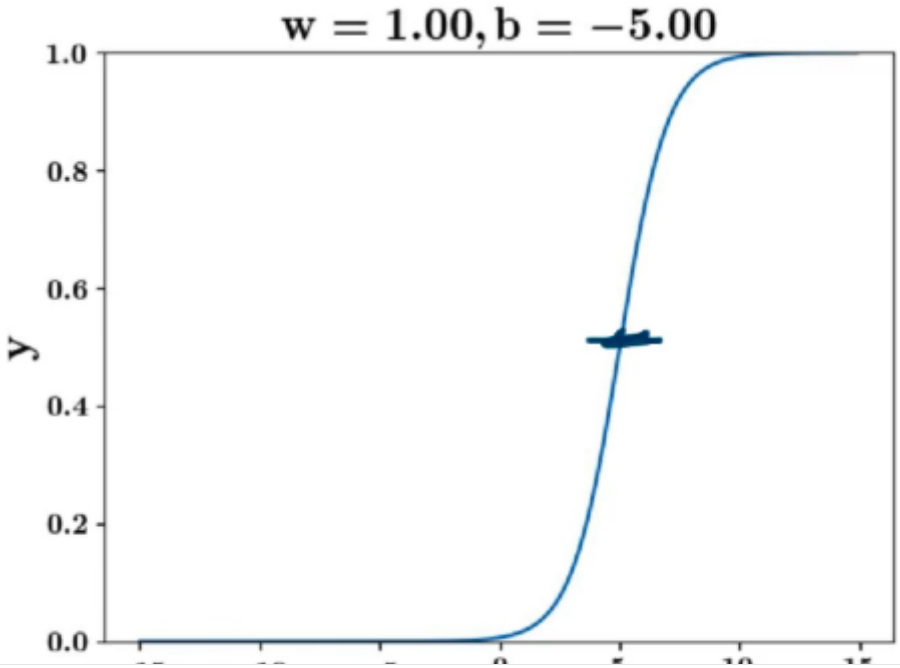

w = 1.00, b = −4.50
1.0
0.8
0.6



And there is an explanation for why this happens:

We know that the value of the sigmoid function would be 0.5 when

$$y = \frac{1}{1+e^{-(wx+b)}} = \frac{1}{2}$$
$$\implies e^{-(wx+b)} = 1$$
$$\implies wx + b = 0$$
$$\implies x = -\frac{b}{w}$$



So, the value of the sigmoid is 0.5 when x is equal to the below:

$$x = -\frac{b}{w}$$

As we keep decreasing b, negative of b would keep on increasing the boundary would shift towards the right(assuming w is positive).

The implication of all these would be when we are minimizing some loss function and change some parameters, we get the idea of how the function plot is going to change.

Sigmoid: Data and Tasks

So far we have looked at MP Neuron and Perceptron model where our **task was of Binary Classification**(output could be 0 or 1) and **we could also use Sigmoid Neuron for this kind of task** with the exception that now instead of getting 0 or 1 as the output, it gives a value between 0 to 1 say 0.7 and we could use that to indicate whether the output is closer to class 1 or class 0. And we can take some threshold value based on the task at the hand to map the output to a particular class for example if the threshold is 0.5 then we can say that it belongs to class 1 and any value less than 0.5 we can map it to class 0.

Of course, once we put a threshold it becomes the same as the dealing with a Perceptron model except that now we have more flexibility.

We could also use this function in the case of Regression task where the output is going to be between 0 to 1.

Data could be a bunch of inputs say 'n' inputs, the **true output is some value between 0 to 1.**

x_1	x_2	y	$\hat{y}$
1	1	0.5	0.6

2	2	0.9	0.5

Sigmoid Loss Function:

We have looked at the 3 jars: Model, Data and Task and we are approximating the relationship between the input and the output using a Sigmoid function.

Now we want to compute the loss given input data, true output, and the Sigmoid function:

x_1	x_2	y
1	1	0.5
2	1	0.8
1	2	0.2
2	2	0.9

Input Data

$$\hat{y} = \frac{1}{1 + e^{-(wx+b)}}$$

We will first compute the predicted output as per the Sigmoid function for the given input data(let's say we have the parameters value, so we will be able to compute the predicted output), once we have the predicted output, we can use the Squared Error Loss function:

x_1	x_2	y	$\hat{y}$

2	1	0.8	0.7
1	2	0.2	0.2
2	2	0.9	0.5

$$Loss = \sum_{i=1}^{4} (y - \hat{y})^2 = 0.18$$

In practice, we might have the true output as Binary and in that case, we could still use the Sigmoid function as the approximation between the input and the output and we could still compute the Loss using the squared error loss:

x_1	x_2	y	$\hat{y}$
1	1	1	0.6
2	1	1	0.7
1	2	0	0.2
2	2	0	0.5

And the point of treating the output as a probability instead of having a real value as the predicted output is that it helps the model to understand which data point is contributing more to the loss and then accordingly adjust its parameters, for example, let's say the true output is 1 for two points and the predicted output is 0.6 and 0.7 for these two data points, then as 0.6 is far from 1 compared to 0.7, 0.6 would contribute more to the loss which would not have been the case for Perceptron model where the predicted output would have been 1 instead of 0.6 and 0.7.

$$Loss = \sum_{i=1}(y - \hat{y})$$

We are now left with 2 jars for the Sigmoid Neuron model which are the Learning Algorithm and the Evaluation metrics which are discussed in this article.

Machine Learning Artificial Intelligence Sigmoid Deep Learning Logistic Sigmoid